COURSE PROJECT

2D Convolution Engine on RTL

Submitted by
PULI SATHYA DHEER SHARMA —
244102408

EEE — VLSI & Nanoelectronics

Contents

Abstract

This project presents the RTL design and verification of a hardware accelerator for two-dimensional (2D) convolution on matrices, a core primitive in image processing and convolutional neural networks (CNNs). The design uses parallel multiply–accumulate units and buffering techniques to perform convolution efficiently in hardware. The implementation was verified using Verilog testbenches and synthesized on FPGA tools.

1 Introduction

2D convolution is a mathematical operation widely used in image processing and neural networks. It forms the backbone of feature extraction in Convolutional Neural Networks (CNNs) and is also used in classical image filtering tasks such as smoothing, sharpening, and edge detection.

1.1 Applications

- Image filtering and enhancement (blurring, sharpening, edge detection).
- Feature extraction in CNNs for computer vision.
- Real-time video processing in embedded and FPGA systems.

2 2D Convolution Algorithm

The 2D convolution of an input image $I(x, y)$ with kernel $K(m, n)$ is defined as:

$$Y(i, j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} K(m, n)\, I(i + m, j + n)$$

where $M \times N$ is the kernel size.

With stride S and padding P, the output size becomes:

$$H_{\text{out}} = \left\lfloor \frac{H - M + 2P}{S} \right\rfloor + 1, \quad W_{\text{out}} = \left\lfloor \frac{W - N + 2P}{S} \right\rfloor + 1$$

3 RTL Implementation

The convolution hardware was designed in Verilog with IEEE floating-point multiplier and adder blocks.

Main Module: conv_2d.v

```
module IEEE #(parameter N=3,M=2)(
    input clk,rst,
    input [7:0] a,b,
    output  reg [7:0] out,
    output reg done
    );
    reg [7:0] A[0:N-1][0:N-1];
    reg [7:0] B[0:M-1][0:M-1];
    reg [31:0]temp1,temp2,p,t1;
    wire [31:0]t2,y;
    reg [3:0] ps,ns;
    localparam init=9,s0=0,s1=1,s2=2,s3=3,s4=4,s5=5,s6
       =6,s7=7,s8=8,s9=9;
    integer c1,c2,c3,c4,c5,c6;
    reg [7:0]sum=0;

    MULTIPLIER m4(temp1,tempe2, t2);
    ADDER a1(t1, t2,y);


    always@(posedge clk)
       begin
          if(rst)
          ps <= init;
          else
          ps<=ns;
      end
   always@(negedge clk)
      begin
        case(ps)
        init: begin
                c1<=0;
                c2<=0;
                c3<=0;
                c4<=0;
                c5<=0;
                c6<=0;
                done<=0;
```

```
38                  end
39          s0:
40               begin
41                  if(c1<M && c2<M)
42                  begin
43                      A[c1][c2] <= a;
44                      B[c1][c2]<=b;
45                  end
46                  else if(M<=c1<N)
47                  A[c1][c2] <= a;
48              end
49          s1:
50              begin
51                  c2<=c2+1;
52              end
53          s2:
54              begin
55                  c2<=0;
56                  c1<=c1+1;
57              end
58
59          s3:
60              begin
61              if(c5<M)
62              begin
63                  if(c6<M)
64                      begin
65                      if(c3 < N - M + 1) begin
66                        if (c4 < N - M + 1) begin
67                          temp1 <= A[c3+c5][c4+c6];
68                          temp2 <= B[c5][c6];
69                        end
70                       end
71                    end
72              end
73              end
74          s4:  begin
75             p <= t2;
76              end
77          s9 : begin
```

```
78              t1 <= y;
79              c6 = c6+1;
80               end
81          s5:
82              begin
83                  c6<=0;
84                  c5<=c5+1;
85              end
86          s6:
87              begin
88
89                  out<=t1;
90                  t1 <=0;
91                  p <=0;
92                  c5<=0;
93                  c6<=0;
94                  sum<=0;
95                  c4<=c4+1;
96              end
97          s7:
98              begin
99                  c4<=0;
100                 c3<=c3+1;
101             end
102         s8:
103             done<=1;
104
105         endcase
106     end
107
108 always@(*)
109  begin
110      case(ps)
111      init: ns=s0;
112      s0: begin
113          if (c1 < N)
114              begin
115                  if(c2<N)
116                      ns = s1;
117                  else
```

```
118                         ns = s2;
119                 end
120             else
121                 ns = s3;
122             end
123         s1:
124             ns=s0;
125         s2:
126             ns=s0;
127         s3: begin
128             if(c3<N-M+1)
129             begin
130                 if(c4<N-M+1)
131                 begin
132                     if (c5<M)
133                         begin
134                             if(c6<M)
135                                 ns = s4;
136                             else
137                                 ns = s5;
138                         end
139                     else
140                         ns = s6;
141                 end
142                 else
143                     ns=s7;
144                 end
145             else
146                 ns=s8;
147             end
148         s4:
149             ns=s9;
150        s9 :
151             ns = s3;
152         s5:
153             ns=s3;
154         s6:
155             ns=s3;
156         s7:
157             ns=s3;
```

```
158         s8:
159             ns=s8;
160         endcase
161     end
162 endmodule
```

Listing 1: Main RTL Module - $conv_2d$

32 bit Multiplier: `mult.v`

```
1
2  module MULTIPLIER #(parameter N=32,M=23,E=8,B=127,O=255)
       ( input [N-1:0]a,b,output reg [N-1:0]out, output reg
       ovrf, reg undrf );
3  reg sign ;
4  reg [E:0]exp ;
5  reg [(2*M+1):0]mant ;
6  reg [M-1:0]N_mant ;
7  reg [E-1:0]R_exp;
8  always@ (*) begin
9   sign = a[N-1]^b[N-1] ;
10  exp = a[N-2:N-E-1] + b[N-2:N-E-1] - B  ;
11  mant = {1'b1,a[M-1:0]} * {1'b1,b[M-1:0]} ;
12             if ( mant[(2*M+1)] == 1 ) begin
13                  N_mant = mant[2*M:M+1] ;
14                  exp = exp + 1 ;
15                  end
16              else begin
17                  N_mant = mant[2*M-1:M] ;
18                  exp = exp ;
19                  end
20              if ( exp > O) begin
21                  R_exp = O ;
22                  N_mant = 0 ;
23                   out = {sign,R_exp,N_mant } ;
24                      ovrf = 1 ;
25                      undrf =0;
26                   end
27             else if ( exp < 0) begin
```

```
                    R_exp = 0;
                    N_mant =0 ;
                    out = {sign,R_exp,N_mant} ;
                    undrf = 1 ;
                    ovrf=0;
                    end
            else
                 begin
                 R_exp = exp[E-1:0] ;
                  ovrf = 0;
                  undrf = 0;
                     out ={sign,R_exp,N_mant} ;
             end

  end


endmodule
```

Listing 2: Testbench Module for 2D Convolution

32 bit Adder: `Add.v`

```

module ADDER_1 (a,b,s,overflow,underflow);
input [31:0]a,b;
output reg [31:0]s;
output reg overflow,underflow;
reg signed [8:0]exp;
reg [23:0]mantisa_a,mantisa_b;
reg [24:0]mantisa_out;
reg [22:0]mantisa_last;

always @(*)
   begin



```

```
17
18            if(a[30:23] >= b[30:23])
19              begin
20                        mantisa_b= {1'b1,b[22:0]} >> (a
                             [30:23]-b[30:23]);
21                        mantisa_a={1'b1,a[22:0]};
22                        exp=a[30:23];
23              end
24            else
25          begin
26                 mantisa_a={1'b1,a[22:0]} >> (b[30:23]-a
                      [30:23]);
27                 mantisa_b={1'b1,b[22:0]};
28                  exp=b[30:23];
29          end
30
31               if(a[31]^b[31])
32                      begin
33                            if(mantisa_a>=mantisa_b)
34                                    begin
35                                    mantisa_out=mantisa_a-
                                         mantisa_b;
36                                    s[31]=a[31];
37                                    end
38                             else
39                                begin
40                                   mantisa_out=mantisa_b-
                                        mantisa_a;
41                                   s[31]=b[31];
42                                end
43                      end //sign comparsion
44                  else  // addition
45                     begin
46                     mantisa_out=mantisa_a+mantisa_b;
47                     s[31]=a[31];
48                      end
49
50
51
52
```

```
53
54
55
56    if(mantisa_out[24]==1)
57       begin
58     mantisa_last=mantisa_out[23:1];
59     exp=exp+1'b00001;
60       end
61
62  else if(mantisa_out[23]==1)
63       begin
64     mantisa_last=mantisa_out[22:0];
65
66       end
67
68   else if(mantisa_out[22]==1)
69       begin
70     mantisa_last={mantisa_out[21:0],1'b0};
71     exp=exp-1;
72       end
73
74   else if(mantisa_out[21]==1)
75       begin
76     mantisa_last={mantisa_out[20:0],2'b00};
77      exp=exp-2;
78       end
79
80       else if(mantisa_out[20]==1)
81       begin
82     mantisa_last={mantisa_out[19:0],3'b000};
83      exp=exp-3;
84       end
85
86    else if(mantisa_out[19]==1)
87       begin
88     mantisa_last={mantisa_out[18:0],4'b0000};
89      exp=exp-4;
90       end
91
92         else if(mantisa_out[18]==1)
```

```
93        begin
94      mantisa_last={mantisa_out[17:0],5'b00000};
95       exp=exp-5;
96        end
97
98          else if(mantisa_out[17]==1)
99        begin
100     mantisa_last={mantisa_out[16:0],6'b000000};
101      exp=exp-6;
102       end
103
104
105         else if(mantisa_out[16]==1)
106       begin
107     mantisa_last={mantisa_out[15:0],7'b0000000};
108      exp=exp-7;
109       end
110
111     else if(mantisa_out[15]==1)
112       begin
113     mantisa_last={mantisa_out[14:0],8'b00000000};
114      exp=exp-8;
115       end
116
117         else if(mantisa_out[14]==1)
118       begin
119     mantisa_last={mantisa_out[13:0],9'b00000000};
120      exp=exp-9;
121       end
122      else if(mantisa_out[13]==1)
123       begin
124     mantisa_last={mantisa_out[12:0],10'b0000000000};
125      exp=exp-10;
126       end
127      else if (mantisa_out[12] == 1)
128      begin
129          mantisa_last = {mantisa_out[11:0], 11'
                b00000000000};
130          exp = exp - 11;
131      end
```

```
132 else if (mantisa_out[11] == 1)
133     begin
134         mantisa_last = {mantisa_out[10:0], 12'
               b000000000000};
135         exp = exp - 12;
136     end
137 else if (mantisa_out[10] == 1)
138     begin
139         mantisa_last = {mantisa_out[9:0], 13'
               b0000000000000};
140         exp = exp - 13;
141     end
142 else if (mantisa_out[9] == 1)
143     begin
144         mantisa_last = {mantisa_out[8:0], 14'
               b00000000000000};
145         exp = exp - 14;
146     end
147 else if (mantisa_out[8] == 1)
148     begin
149         mantisa_last = {mantisa_out[7:0], 15'
               b000000000000000};
150         exp = exp - 15;
151     end
152 else if (mantisa_out[7] == 1)
153     begin
154         mantisa_last = {mantisa_out[6:0], 16'
               b0000000000000000};
155         exp = exp - 16;
156     end
157 else if (mantisa_out[6] == 1)
158     begin
159         mantisa_last = {mantisa_out[5:0], 17'
               b00000000000000000};
160         exp = exp - 17;
161     end
162 else if (mantisa_out[5] == 1)
163     begin
164         mantisa_last = {mantisa_out[4:0], 18'
               b000000000000000000};
```

```
165         exp = exp - 18;
166     end
167 else if (mantisa_out[4] == 1)
168     begin
169         mantisa_last = {mantisa_out[3:0], 19'
               b00000000000000000000};
170         exp = exp - 19;
171     end
172 else if (mantisa_out[3] == 1)
173     begin
174         mantisa_last = {mantisa_out[2:0], 20'
               b00000000000000000000};
175         exp = exp - 20;
176     end
177 else if (mantisa_out[2] == 1)
178     begin
179         mantisa_last = {mantisa_out[1:0], 21'
               b00000000000000000000};
180         exp = exp - 21;
181     end
182     else if (mantisa_out[1] == 1)
183     begin
184         mantisa_last = {mantisa_out[0], 22'
               b00000000000000000000};
185         exp = exp - 21;
186     end
187
188           else
189           begin
190          mantisa_last=0;
191           end
192
193
194
195 if(exp>254)
196 begin
197 overflow=1;
198 underflow=0;
199 s[30:0]={8'hff,mantisa_last};
200 end
```

```
201 else if(exp<0)
202 begin
203 overflow=0;
204 underflow=1;
205 s[30:0]={8'b00000000,mantisa_last};
206 end
207 else
208 begin
209 overflow=0;
210 underflow=0;
211 s[30:0]={exp[7:0],mantisa_last};
212 end
213
214
215    end
216 endmodule
```

Listing 3: Testbench Module for 2D Convolution

Constraint File cnstrnt.v

```
1  # Create a virtual clock with a T ns period where clk is
       initialized
2  create_clock -period 20.000 -name virtual -waveform
       {0.000 10.000} -add [get_ports clk]
3
4
5  # Set correct input delays (min should be <= max)
6  set_input_delay -clock virtual -min 2.2 [get_ports a]
7  set_input_delay -clock virtual -max 3.3 [get_ports a]
8  set_input_delay -clock virtual -min 2.2 [get_ports b]
9  set_input_delay -clock virtual -max 3.3 [get_ports b]
10
11 # Set correct output delays (min should be <= max)
12 set_output_delay -clock virtual -min 0.2 [get_ports out]
13 set_output_delay -clock virtual -max 0.9 [get_ports out]
14 set_output_delay -clock virtual -min 0.2 [get_ports done
       ]
```

```
set_output_delay -clock virtual -max 0.9 [get_ports done
   ]
```

Listing 4: Testbench Module for 2D Convolution

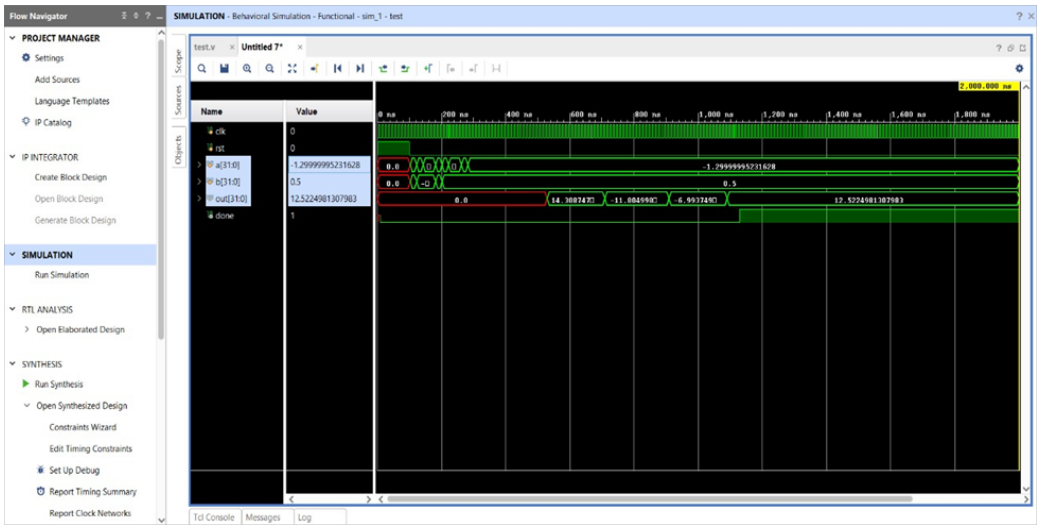
Testbench: `test.v`

```
// Revision 0.01 - File Created
module test();
    reg clk, rst;
    reg [31:0] a, b;
    wire [31:0] out;
    wire done;

    // Instantiate the convolution module
    IEEE #(3,2) a1(clk,rst,a,b,out,done);

    always #5 clk = ~clk;

    initial begin
        clk = 0;
        rst = 1;
        #100;

        rst = 0;

        #3  a = 32'h3F99999A; b=32'h3F000000;
        #20 a = 32'hC0200000; b = 32'hBFF19999;
        #20 a = 32'h4059999A;
        #40 a = 32'hBF4CCCCD; b = 32'hBF666666;
        #20 a = 32'h40066666; b = 32'h40133333;
        #20 a = 32'hC0733333;
        #40 a = 32'h3F19999A;
        #20 a = 32'hBFA66666;
        #20 a = 32'h40366666;
    end
endmodule
```

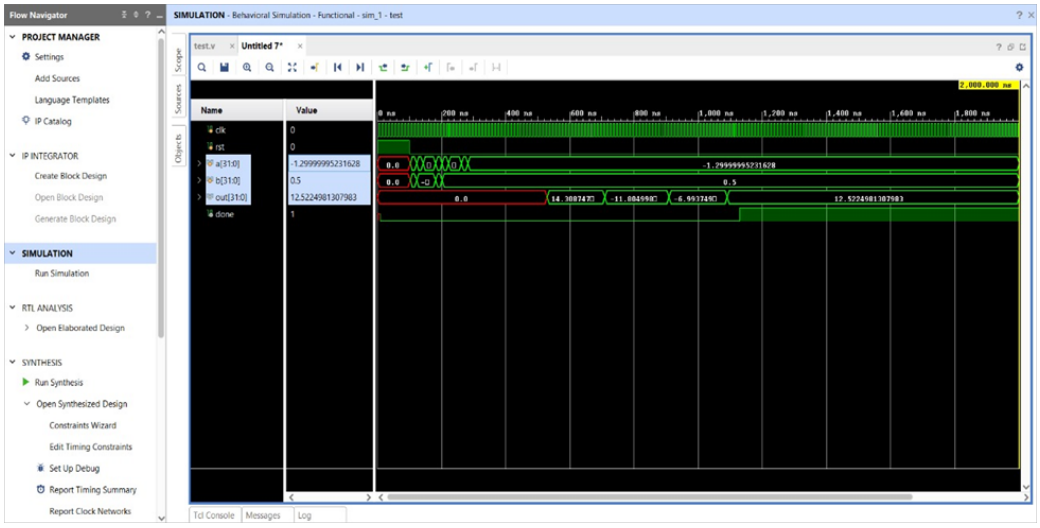
Listing 5: Testbench Module for 2D Convolution

4 Results

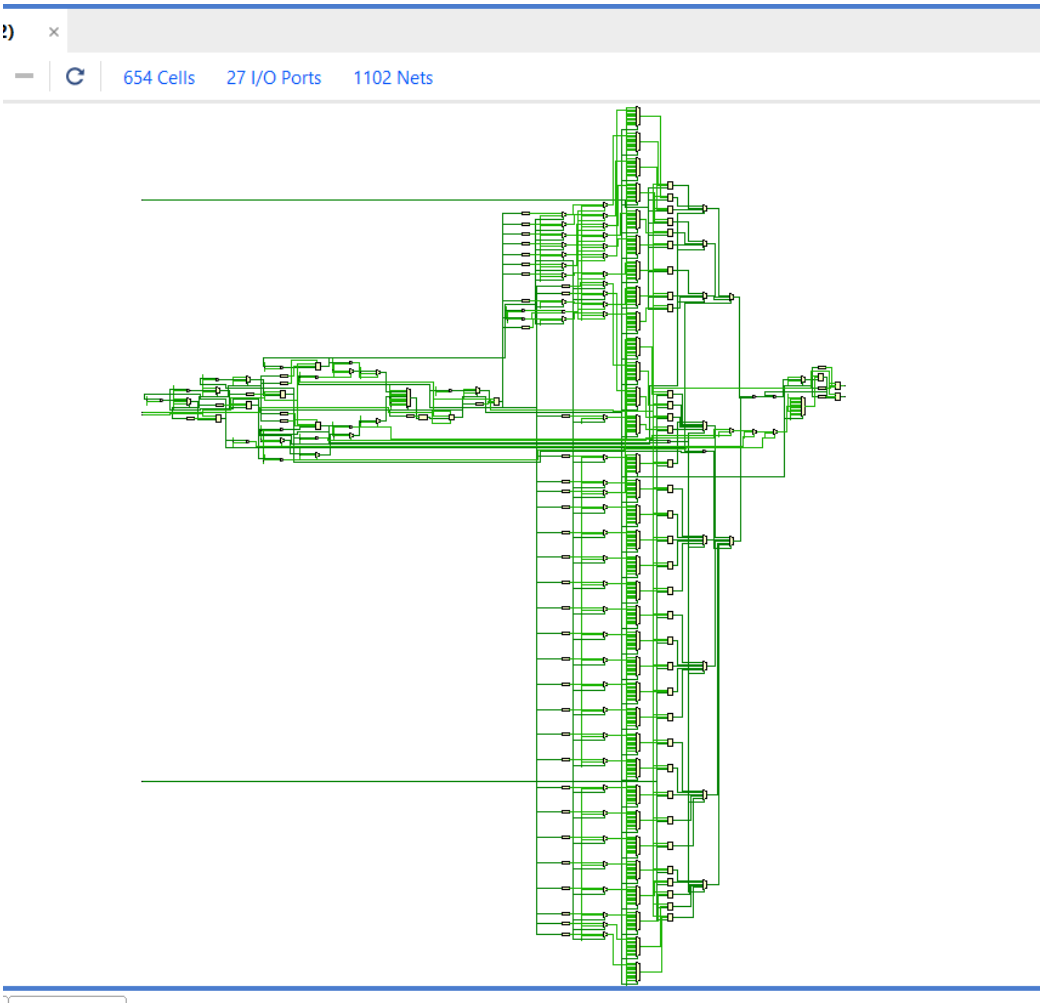
4.1 Behavioural Simulation



4.2 Post Synthesis Functional Simulation



4.3 Elaborated Design



4.4 Utilization

Hierarchy

Name	Slice LUTs (41000)	Slice Registers (82000)	Bonded IOB (300)	BUFGCTRL (32)
twoD_con	413	489	27	1

5 Conclusion

The RTL design of a 2D convolution engine was successfully implemented and verified. The results show correct functionality across simulation and synthesis. This accelerator demonstrates the feasibility of performing CNN-like operations in FPGA/ASIC hardware for real-time image processing.